Finding Happiness: An Analysis of the Maximum Happy Vertices Problem

Lewis, R.[1], Thiruvady, D.[2], and Morgan, K.[3]

[1]School of Mathematics, Cardiff University, Cardiff, CF24 4AG, Wales.
[2]School of Mathematical Sciences, Monash University, Clayton VIC, Australia.
[3]School of Information Technology, Deakin University, Geelong VIC, Australia.,
LewisR9@cf.ac.uk, dhananjay.thiruvady@monash.edu, kerri.morgan@deakin.edu.au

November 15, 2018

Abstract

The maximum happy vertices problem involves determining a vertex colouring of a graph such that the number of vertices assigned to the same colour as all of their neighbours is maximised. This problem is trivial if no vertices are precoloured, though in general it is NP-hard. In this paper we derive a number of upper and lower bounds on the number of happy vertices that are achievable in a graph and then demonstrate how certain problem instances can be broken up into smaller subproblems. Four different algorithms are also used to investigate the factors that make some problem instances more difficult to solve than others. In general, we find that the most difficult problems are those with relatively few edges and/or a small number of precoloured vertices. Ideas for future research are also discussed.

Keywords: Happy Colouring; Graph Colouring; Problem Subdivision; Integer Programming.

1 Introduction

Problems that involve the colouring of graphs come in many guises including vertex colouring, edge colouring, face colouring, precolouring, list colouring, dynamic colouring, and equitable colouring [12]. Such problems typically involve assigning colours to particular elements of a graph such that elements deemed to be "conflicting" are given different colours. Perhaps the most well-known of these problems is the vertex colouring problem where, using a limited set of colours, each vertex of a graph needs to be assigned to a colour such that no pair of adjacent (conflicting) vertices are given the same colour. This has applications in many timetabling, logistical and scheduling problems, where sets of conflicting entities such as tasks, events, or people need to be efficiently allocated to a limited set of resources [9, 12, 14, 16].

In recent times, interest has been growing in graph colouring problems where adjacent entities need to be assigned to the *same* colour, as opposed to different colours [3, 5, 15, 17, 18]. In the case of vertex colouring, this can be useful in areas such as social networking, where we might be interested in assigning groups of related people (vertices) to the same resource, such as a team or a task. Recent studies have also looked at how we might design seating plans for large social gatherings such as weddings, where the aim is simultaneously place people with their friends, but apart from their adversaries [7, 13].

In a recent paper, Li and Zhang [15] introduced the notion of vertex "happiness". Let $\Gamma(v)$ denote the set of vertices adjacent to a vertex v.

Definition 1. *Let $G = (V, E)$ be a simple graph and let $c : V \to \{1, \ldots, k\}$ be a colouring of all vertices in G. A vertex $u \in V$ is* happy *if $c(u) = c(v)$ for all $v \in \Gamma(u)$; else it is* unhappy.

In other words, a vertex is considered happy if and only if it is assigned to the same colour as all of its neighbouring vertices, otherwise it is unhappy. This naturally gives rise to the following optimisation problem:

Definition 2. *The* Maximum Happy Vertices *(MHV) problem takes an undirected graph $G = (V, E)$ with n vertices and m edges, an integer k, a subset of vertices $V' \subseteq V$ where $|V'| \geq k$, and a partial colouring $c : V' \to \{1, \ldots, k\}$ such that $\forall i \in \{1, \ldots, k\}$, $\exists v \in V' : c(v) = i$. The goal is to extend c to a total colouring $\tilde{c} : V \to \{1, \ldots, k\}$ such that the number of happy vertices is maximised.*

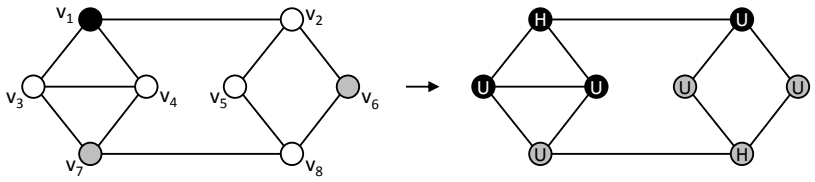

Figure 1: Example problem instance with $n = 8$ vertices, $m = 11$ edges, $k = 2$ colours, and $|V'| = 3$ precoloured vertices (left). The graph on the right shows a corresponding solution, defined by the total colouring $\tilde{c} : V \to \{1, 2\}$, comprising $H(G) = 2$ happy vertices and $n - H(G) = 6$ unhappy vertices.

In this definition, the vertices contained in V' are the problem's *precoloured* vertices; otherwise they are *free* vertices, belonging to the set $V - V'$. Free vertices are originally uncoloured, but will all need to be coloured in the final solution.

As stated in Definition 2, it is necessary that each colour is used at least once in the partial colouring c. Precoloured vertices can themselves be happy or unhappy—indeed, their status could even be determined by c before any of the free vertices have been coloured. Once a total colouring of G has been defined, the number of happy vertices is denoted by $H(G)$ where, $0 \leq H(G) \leq n$. The optimum (maximum) number of happy vertices in a graph G is denoted by $H(G)^*$.

An example problem instance and candidate solution to the MHV problem is shown in Figure 1. This illustrates that the problem can also be seen as one of partitioning: as input, we take a set of nonempty colour classes $\{V_1, V_2, \ldots, V_k\}$ such that $\bigcup_{i=1}^{k} V_i = V'$ and $v \in V_i \Leftrightarrow c(v) = i$. Our task is to assign the free vertices to the k colour classes such that $H(G)$ is maximised, $\bigcup_{i=1}^{k} V_i = V$, and $V_i \cap V_j = \emptyset \; \forall i \neq j$. The example solution in Figure 1 may therefore be written as $\{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}\}$.

A practical application of the MHV problem can occur where we have a set people, some of whom have been preassigned to groups for a team building exercise. The remaining people then also need to be assigned to these groups such that the maximum number of people are happy (i.e., in a group containing all of their friends). Similarly, it could arise when seeking to assign these people to dormitories or hotel rooms. The problem also has more general applications in cluster analysis, where some objects are preassigned to clusters, and the task is to assign the remaining objects to clusters such that strongly related objects occur in the same cluster [10]. Let us note, though, that although this problem may somehow sound like the "opposite" of a vertex colouring problem, we cannot tackle it by simply producing a proper colouring of the complement of G, as this simply leads to a solution in which nonadjacent pairs of vertices are never assigned to the same colour.

As mentioned, the MHV problem was originally proposed by Li and Zhang in 2015 [15]. In their paper, the authors also considered the related *Maximum Happy Edge* (MHE) problem, which involves colouring vertices of a graph such that the number of "happy edges" (i.e., edges with endpoints of the same colour) is maximised. They first showed that the MHE problem with $k \geq 3$ is NP-complete by demonstrating a polynomial reduction from the three terminal cut problem. The MHV problem was itself then shown to be NP-complete by showing that MHE $\propto$ MHV. It was also shown that both problems are polynomially solvable for $k \leq 2$.

In subsequent work, Aravind et al. [4] went on to show that the MHE and MHV problems are polynomially solvable for cycle-free graphs (trees). Later, Aravind et al. [5] proved that both problems also remain NP-complete for general bipartite graphs (for $k \geq 3$) and that the MHV problem is hard for split graphs. It is now also known that both problems are polynomially solvable when G has bounded treewidth or bounded neighbourhood diversity [3, 5].

To our knowledge, three approximation algorithms for the MHV problem on general graphs have previously been suggested in the literature. In their original paper, Li and Zhang [15] proposed two single-parse constructive algorithms for the problem with approximation ratios of $1/k$ and $\Omega(1/\Delta^3)$. These are considered in more detail in Section 2. More recently, Zhang et al. [18] presented an algorithm with an approximation ratio of $1/(\Delta + 1)$ that operates by solving a simple linear programming relaxation of the MHV problem. If the solution produced for this relaxation happens to be integral, then it corresponds to the optimum solution; more likely, however, is that the solution is fractional, in which case the non-integer decision variables are rounded via an iterative process to produce an approximate solution.

Given a total colouring of a graph, observe that the removal of an edge can either maintain or increase the current number of happy vertices, but that the addition of an edge can only maintain or decrease the number of happy vertices. Intuitively, this suggests that optimum solutions for dense graphs will tend to feature fewer happy vertices than those of sparse graphs. In this paper we confirm this by taking the linear programming ideas of Zhang et al. [18] a step further by implementing and testing a more compact integer programming (IP) formulation for the MHV problem. In Section 2 we start by reviewing the two constructive algorithms for the MHV problem. In Section 3 we then use these to help derive a number of bounds for the problem, before describing a way of subdividing problem instances in Section 4. Section 5 then contains our IP formulation together with results from a large set of experiments that demonstrate the characteristics that contribute towards making a problem difficult to solve. A CMSA algorithm that

extends this IP formulation is then also proposed to help investigate the affects of problem size. Section 6 concludes the paper and conducts further discussions.

2 Constructive Algorithms

As mentioned, Li and Zhang [15] previously proposed two single-parse constructive algorithms for the MHV problem. In this section we now review these in detail, particularly because they will be used as a point of comparison later in this paper. The first algorithm, GREEDY-MHV, operates by simply assigning all free vertices to colour 1 before calculating the resultant number of happy vertices. This is then repeated using colours $2, \ldots, k$, and the best of these k solutions is taken. This algorithm has an approximation ratio $1/k$ and complexity $O(km)$ since, for each colour $1, \ldots, k$, it is necessary to determine the happiness of each free vertex by inspecting all of its neighbours.

The second algorithm from [15] is the so-called subset-growth algorithm (GROWTH-MHV), which is known to have an approximation ratio within a factor of $\Omega(1/\Delta^3)$, where $\Delta = \max\{\deg(v) : v \in V\}$ is the maximum vertex degree in the graph being considered. This algorithm involves using vertex labels that are based on whether the vertices are coloured or not and also the statuses of their neighbours. These labels, which we now define, are most conveniently described using the partition-based interpretation of the MHV problem (Section 1), where vertices need to be assigned to the colour classes $V_1, \ldots, V_k$.

Definition 3. *Let v be a vertex that has been assigned to a colour class $V_i \in \{V_1, \ldots, V_k\}$:*

- *v is an* H-vertex *if it is happy (i.e., all neighbours of v are also assigned to V_i).*
- *v is a* U-vertex *if it is destined to be unhappy (i.e., at least one neighbour of v has been assigned to a colour class $V_j \neq V_i$).*
- *v is a* P-vertex *if it has the potential to be happy (i.e., some neighbours of v are uncoloured, but all of its coloured neighbours are in V_i).*

Now let v be a vertex not yet assigned to a colour class:

- *v is an* LP-vertex *if it is adjacent to a P-vertex.*
- *v is an* LH-vertex *if it is not adjacent to a P-vertex, but has the potential to become happy (i.e., it is adjacent to U-vertices of only one colour).*
- *v is an* LU-vertex *if it is not adjacent to a P-vertex, and it is destined to be unhappy.*
- *v is an* LF-vertex *if it is not adjacent to any coloured vertex.*

The GROWTH-MHV algorithm operates by colouring one or more free vertices at each iteration. Specifically, the neighbours of P-vertices are prioritised, followed by LH-vertices and their neighbours, then LU-vertices, and then LF vertices. It executes as follows.

1. Let v be a P-vertex. If no such vertex exists, go to Step 2. Else, let i be the colour of v, assign all neighbours of v to V_i and return to Step 1.
2. Let v be an LH-vertex. If no such vertex exists, go to Step 3. Else, let i be the colour of any U-vertex adjacent to v, assign v and all its uncoloured neighbours to V_i, and go to Step 1.
3. Let v be an LU-vertex. If no such vertex exists, go to Step 4. Else, let i be the colour of any U-vertex adjacent to v, assign v to V_i, and return to Step 1.
4. Let v be an LF-vertex. If no such vertex exists, all vertices have been coloured, so end. Else, assign v to an arbitrary colour and return to Step 1.

Note that in Steps 1 and 2 of this algorithm the selected vertex v is guaranteed to become happy. On completion of any of the steps, the labels of various other vertices in the graph can also change. For Steps 3 and 4, this could be v and any of its neighbours; for Steps 1 and 2, it could be any vertex within a distance of three (edges) from v.[1] Since such updates are of complexity $O(m)$, the overall complexity of GROWTH-MHV is $O(nm)$.

3 Bounds for the MHV Problem

In this section we give some bounds on the number of happy vertices that are achievable for various graphs. Lower bounds are derived by considering the behaviour of the GREEDY-MHV algorithm and a simplified variant, while upper bounds are generated using the concept of *unhappy paths*, given in Definition 5 below.

[1] In [15], Step 1 and 2's affected vertices are stated to be anything within distance two of v. However, this is incorrect. For instance, consider the path of vertices (v_1, v_2, v_3, v_4) with labels (P, LP, P, LP), and colours (red, none, blue, none), respectively. If v_1 is now selected by the algorithm, this causes v_2 to also become red; consequently, v_1 is now an H-vertex, and v_2 a U-vertex. In addition, v_3 now becomes a U-vertex, which could also change the label of v_4 to either LH or LU.

3.1 Lower Bounds

For lower bounds, we first consider *d-regular* graphs (that is, graphs in which every vertex has degree of exactly d).

Theorem 1. *Let $G = (V, E)$ be a d-regular graph and $V' \subseteq V$ be the precoloured vertices. Then there exists a solution with at least $n - |V'|(d+1)$ happy vertices.*

Proof. According to the behaviour of GREEDY-MHV, all vertices in $(V - V')$ will be assigned to a single colour. This means that, at most, all of the vertices of V' and their neighbours $\Gamma(V')$ are unhappy, so

$$\begin{aligned} H(G) &= n - \text{the number of unhappy vertices} \\ &\geq n - |V'| - |\Gamma(V')| \\ &\geq n - |V'| - |V'|d \\ &= n - |V'|(d+1). \end{aligned} \tag{1}$$

□

A similar theorem generalises this result to all graphs by simply replacing d with a graph's maximum degree Δ:

Theorem 2. *Given $G = (V, E)$ and $V' \subseteq V$, there exists a solution with at least $n - |V'|(\Delta + 1)$ happy vertices.*

Note, however, that this bound will be lower and potentially less accurate for graphs whose degree distributions have a positive skew—that is, a relatively small number of high-degree outliers—such as scale-free graphs.

We now make some statements concerning random graphs. These are graphs in which each pair of vertices u, v is connected by an edge with a fixed probability p. The set of all such graphs for particular p and n values is denoted by $\mathcal{G}(n, p)$. Given a particular problem instance, let $A \subseteq V'$ be the largest subset of precoloured vertices that are assigned to the same colour $a \in \{1, \ldots, k\}$. Also, let $H(G)^a$ denote the number of happy vertices that would result if all free vertices of G were also assigned to colour a.

Theorem 3. *Let $G \in \mathcal{G}(n, p)$. Then:*

$$(n - \frac{(k-1)|V'|}{k})(1-p)^{\frac{|V'|(k-1)}{k}} \leq \mathrm{E}[H(G)^a] \leq (n - \frac{(k-1)|V'|}{k})(1-p)^{\frac{|V'|(k-1)}{k}} + \frac{|V'|(k-1)}{k}. \tag{2}$$

Proof. Observe that $|V'|/k \leq |A| \leq |V'| - k + 1$. If all free vertices are assigned to colour a, it follows that a vertex in the set $V - (V' - A)$ is happy if and only if it has no neighbours belonging to the set $V' - A$. Without loss of generality, suppose $v_1, v_2, \ldots, v_l$ are the vertices assigned to colour a (i.e., belonging to the set $V - (V' - A)$). Now let X_i be a binary indicator variable for the event that vertex v_i is happy (for $i \in \{1, 2, \ldots, l\}$). Then the expected number of happy vertices among these is:

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{l} X_i\right] \\ &= \sum_{i=1}^{l} \mathrm{E}[X_i] \text{ (by linearity of expectation)} \\ &= \sum_{i=1}^{l} (1-p)^{|V'-A|} \\ &= (n - |V'| + |A|)(1-p)^{|V'-A|} \end{aligned} \tag{3}$$

Now, because $|A| \geq |V'|/k$, Equation (3) becomes:

$$\begin{aligned} \mathrm{E}[X] &\geq (n - \frac{(k-1)|V'|}{k})(1-p)^{|V'-A|} \\ &\geq (n - \frac{(k-1)|V'|}{k})(1-p)^{\frac{|V'|(k-1)}{k}}. \end{aligned} \tag{4}$$

The variance of X is:

$$\begin{aligned} \mathrm{Var}[X] &= \sum_{i=1}^{l} \mathrm{Var}[X_i] \\ &= \sum_{i=1}^{l} \left(\mathrm{E}[X_i^2] - \mathrm{E}[X_i]^2\right) \\ &= l\left((1-p)^{|V'-A|} - (1-p)^{2|V'-A|}\right) \\ &= (n - |V'| + |A|)(1-p)^{|V'-A|}\left(1 - (1-p)^{|V'-A|}\right). \end{aligned} \tag{5}$$

Again, since $|A| \geq |V'|/k$, Equation (5) becomes:

$$\begin{aligned}\text{Var}[X] &\geq (n - \frac{(k-1)|V'|}{k})(1-p)^{|V'-A|}\left(1-(1-p)^{|V'-A|}\right) \\ &\geq (n - \frac{(k-1)|V'|}{k})(1-p)^{\frac{|V'|(k-1)}{k}}\left(1-(1-p)^{|V'-A|}\right). \end{aligned} \tag{6}$$

Here, note that there will be at least $k-1$ vertices in the set $V'-A$, meaning that $(1-p)^{|V'-A|} \geq (1-p)^{k+1}$. It follows that:

$$\text{Var}[X] \geq (n - \frac{(k-1)|V'|}{k})(1-p)^{\frac{(k-1)|V'|}{k}}\left(1-(1-p)^{k+1}\right). \tag{7}$$

Finally, it is also possible that some of the vertices in the set $V'-A$ will be happy. Thus, the expected number of happy vertices, $E(H(G)^a)$ lies in the range:

$$\begin{aligned}(n - \frac{(k-1)|V'|}{k})(1-p)^{\frac{|V'|(k-1)}{k}} \leq \text{E}[H(G)^a] &\leq (n - \frac{(k-1)|V'|}{k})(1-p)^{\frac{|V'|(k-1)}{k}} + |V'-A| \\ &\leq (n - \frac{(k-1)|V'|}{k})(1-p)^{\frac{|V'|(k-1)}{k}} + \frac{|V'|(k-1)}{k} \end{aligned} \tag{8}$$

as required. □

Here, the upper bound is reached when all vertices that are not coloured with colour a are happy. When $k=1$, the expected number of happy vertices will be n, which is confirmed by Theorem 3 since $(n+0)(1-p)^0 = n \leq E[H(G)^a] \leq n(1-p)^0 + 0 = n$. On the other hand, for fixed $k > 1$, as $|V'| \to n$, the bounds for the expected number of happy vertices are $(n - \frac{n(k-1)}{k})(1-p)^{\frac{n(k-1)}{k}} = (1-p)^{\frac{n(k-1)}{k}} \leq \text{E}[H(G)^a] \leq (1-p)^{\frac{n(k-1)}{k}} + \frac{n(k-1)}{k}$.

3.2 Upper Bounds

To generate upper bounds, it is useful to first consider the conditions necessary for *all* vertices in a graph to be happy. We can then use these ideas to provide a method of calculating an upper bound on the number of happy vertices in any arbitrary graph.

Definition 4. *Let G be a graph comprising components $C_1, \ldots, C_l$. Then* $\text{col}(C_i)$ *denotes the number of different colours used by the precoloured vertices in a component C_i.*

Obviously, if G is connected, then it contains just one component C_1, giving $\text{col}(C_1) = \text{col}(G) = k$.

Definition 5. *Given a graph $G = (V, E)$, a subset of vertices $V' \subseteq V$, and a partial colouring $c : V' \to \{1, \ldots, k\}$, an* unhappy path *is a simple path whose internal vertices (if any) are not coloured, and whose terminal vertices u and v are precoloured differently: i.e., $c(u) \neq c(v)$.*

Example unhappy paths from Figure 1(a) include (v_1, v_2, v_6) and $(v_1, v_2, v_5, v_8, v_7)$, with lengths of two and four (edges) respectively. These definitions bring us to the following:

Theorem 4. *$H(G)^* = n$ if and only if G comprises $l \geq k$ components $C_1, \ldots, C_l$, and $\text{col}(C_i) \leq 1$ for all $i \in \{1, \ldots, l\}$.*

Proof. If $H(G) = n$, then a total colouring has been achieved in which the vertices in all components are happy. This implies that each component C_i has either $\text{col}(C_i) = 0$, in which case all vertices in C_i have been assigned to the same arbitrary colour, or $\text{col}(C_i) = 1$, in which case each free vertex in C_i has been allocated to the same colour as its precoloured vertices.

Now let C_i be a component for which $\text{col}(C_i) > 1$. This implies the existence of an unhappy path in C_i. If this path is of length one, neither u nor v can be happy. Similarly, if its length is greater than one, then in a total colouring there must exist at least one pair of adjacent vertices in this path whose colours are different. Hence $H(G)^* < n$. □

The above theorem also allows us to make statements about the behaviour of the GROWTH-MHV algorithm.

Theorem 5. *If $H(G)^* = n$ then* GROWTH-MHV *is exact.*

Proof. It is sufficient to consider each component C_i of G separately. According to Theorem 4, each C_i has either $\text{col}(C_i) = 0$ or $\text{col}(C_i) = 1$. If $\text{col}(C_i) = 1$, then any coloured vertex v in C_i that is not yet happy will be a P-vertex and will therefore be selected in Step 1 of the algorithm. All neighbouring vertices will then assume the same colour as v and will each become either an H-vertex or a P-vertex. Step 1 will be repeated on C_i until all of its vertices are H-vertices.

If $\text{col}(C_i) = 0$, then C_i comprises only LF-vertices. Step 4 of the algorithm will now select one of these and assign it to a random colour, resulting in $\text{col}(C_i) = 1$. The previous case now applies. □

Because their presence in a graph implies the existence of unhappy vertices, unhappy paths can also be used for generating an upper bound $\bar{H}(G)$ on $H(G)^*$. A process for doing this is outlined in the following steps. It involves repeatedly identifying and removing unhappy paths from a graph while keeping count of the minimum possible number of unhappy vertices x. To start, x is set to zero, and all precoloured vertices are marked as having not yet been counted.

1. Let $P = (u, v)$ be an unhappy path in G of length one. If no such path exists, go to Step 3.
2. If neither u nor v have yet been counted, set $x = x+2$; else, if just one of u or v has been counted, set $x = x+1$. Now, remove the edge $\{u, v\}$ from G, mark u and v as having been counted, and return to Step 1.
3. Let P be an unhappy path in G with length greater than one. If no such path exists, return the upper bound $\bar{H}(G) = n - x$ and end.
4. Let u and v be the terminal vertices of P. If neither u nor v have yet been counted, set $x = x + 2$; else, set $x = x + 1$. Now, mark u and v as having been counted, remove all internal vertices in P from G, and return to Step 3.

Note that this process involves mechanisms in Steps 2 and 4 to avoid counting vertices more than once in the calculation of x. In the first two steps, all unhappy paths of length one are removed from G and x is incremented accordingly. Steps 3 and 4 then carry out a similar process by removing longer paths, though the rules for incrementing x are slightly different to allow the possibility of multiple unhappy paths existing between two differently-precoloured vertices.

In Step 3, any arbitrary unhappy path P can be chosen. For the results reported in this paper we use a greedy strategy of selecting the shortest unhappy path in the current graph, which is identified through applications of breadth-first search. At each step this results in the smallest number of vertices being removed from the current graph. The intention is to allow the process to iterate for a larger number of cycles, thereby increasing x and improving $\bar{H}(G)$. Other strategies might also be applied in practice, however.

4 Problem Subdivision

In this section we show how instances of the happy colouring problem can be broken up into smaller components, allowing the algorithm of choice to be applied to each component separately. These sub-solutions can then be merged into a single solution for the entire graph.

Firstly, observe that if we have a disconnected graph G comprising multiple components $C_1, \ldots, C_l$ then the happiness (or otherwise) of a vertex in one component can have no influence on the status of vertices in another component. This means that if we were to compute total colourings for all components separately, then $\sum_{i=1}^{l} H(C_i) = H(G)$. We now extend this idea to connected graphs using the idea of *H-U separating sets*.

Definition 6. *Given a graph G, a* separating set *S is a subset of vertices $S \subseteq V$ whose removal increases the number of components in G. An* H-U separating set *is a separating set comprising only H- and U-vertices (recalling Definition 3).*

Let $C_1, \ldots, C_{l>1}$ be the components resulting from the removal of an H-U separating set S. Now, for each $C_i \in \{C_1, \ldots C_l\}$, let G_i be the subgraph induced by the vertices of C_i and S. Finally, if G_i contains a vertex $v \in S$ that was a U-vertex in G but is now an H-vertex, add an additional dummy vertex u with an arbitrary colour $c(u) \neq c(v)$ together with the edge $\{u, v\}$. This ensures that v is also a U-vertex in G_i.

An example of this process is shown in Figures 2(a) and (b), resulting in the subgraphs G_1, G_2, and G_3. Note that two dummy vertices marked by asterisks have been imposed here, as required. Figure 2(c) then shows total colourings of each subgraph, produced by an arbitrary method. A final solution to the original graph G is formed by merging these subgraphs and deleting the dummy vertices, as shown in Figure 2(d).

Theorem 6. *Let $G_1, \ldots, G_l$ be totally coloured subgraphs that were originally constructed using an H-U separating set S, as explained above. Also, let G be a totally coloured graph, formed by merging $G_1, \ldots, G_l$. Then*

$$H(G) = \left(\sum_{i=1}^{l} H(G_i) \right) - (l-1)H(S). \tag{9}$$

Proof. It is sufficient to show that a vertex v is happy in G_i if and only if it is also happy in G.

First, note that in each G_i, vertices originating from S must have the same status as their counterparts in G: if $v \in S$ is an H-vertex in G, then its set of neighbours in G_i are a subset of its neighbours in G, hence it is also an H-vertex in G_i; similarly, if $v \in S$ is a U-vertex in G, then the possible introduction of dummy vertices also guarantees it is a U-vertex in G_i. We only need to consider vertices from the set $V(G) - S$ in G and $V(C_i)$ in G_i.

The proof is now trivial since the neighbours of a vertex $v \in V(C_i)$ are either themselves in C_i, or are members of S and therefore have their statuses fixed as either U or H; hence the happiness of v has no effect on the happiness of any vertex outside of C_i.

Finally, the subtraction of $(l-1)H(S)$ is necessary to ensure that H-vertices in S are only counted once when the subgraphs are combined to form G. □

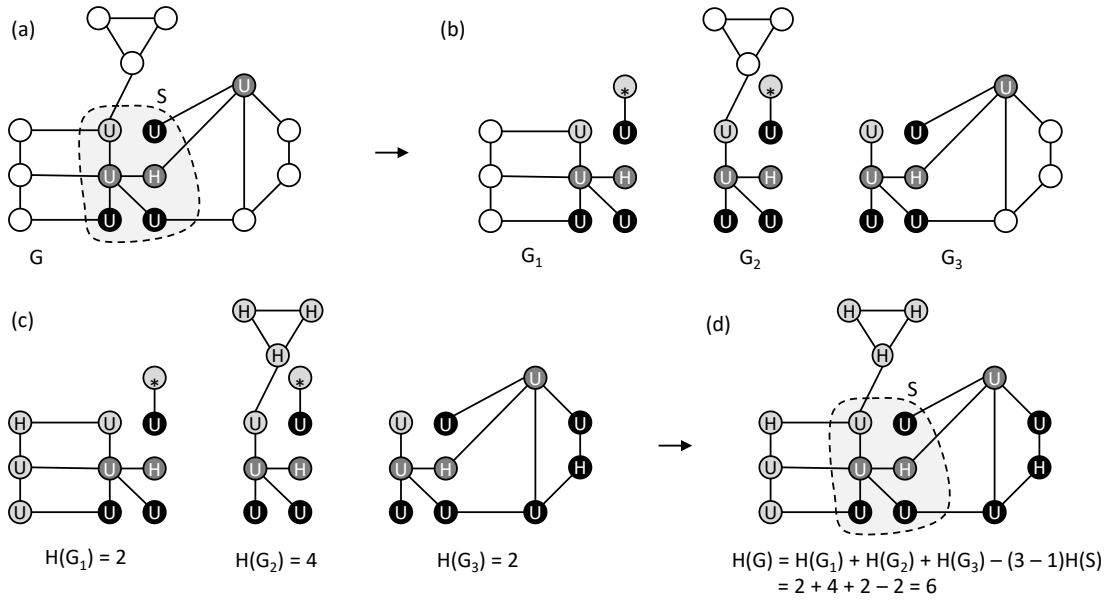

Figure 2: Example of how a graph G can be subdivided using an H-U separating set S. Additional vertices that occur during the creation of G_1 and G_2 are marked by asterisks.

An immediate corollary of Theorem 6 is that $\sum_{i=1}^{l} H(G_i)^* - (l-1)H(S) = H(G)^*$. That is, if optimum solutions can be found for each subgraph G_i, then the merged solution for G will also be optimum. If we are lucky, some of the extracted subgraphs G_i might also turn out to be polynomially solvable, such as when $\text{col}(G_i) \leq 2$, or when G_i is a tree.

The problem of identifying separating sets in graphs is readily solvable by various polynomial-time algorithms (e.g., [11]), and it only takes the addition of a simple checking step to determine whether each separating set also comprises just H- and U-vertices. In practice, if a graph contains no H-U separating set, it would also be possible to colour some free vertices to help produce one; however, if suboptimal choices are made in these assignments, this will also prohibit the final merged solution from being optimum.

5 Experimental Analysis

In this section we now conduct an empirical analysis of the MHV problem. In particular, we use an IP formulation together with the GREEDY-MHV and GROWTH-MHV algorithms to identify features that make instances of this problem difficult to solve. In addition, Section 5.4 also proposes a CMSA algorithm for tackling larger instances of this problem. All experiments reported here were performed on single-threaded 2.80 GHz processors using 5 GB RAM. Copies of our source code and problem instance generator can be found at [1, 2].

5.1 Set-up

For our experiments both the GREEDY-MHV and GROWTH-MHV algorithms were implemented in C++. In the case of GROWTH-MHV, when multiple vertices were available for selection in a particular step, ties were broken by selecting the vertex with the highest degree. This tended to give shorter run times because, for Steps 1 and 2, more neighbouring vertices would be coloured, resulting in fewer iterations of the overall algorithm.

Our IP method is based on the following formulation, which we implemented using Gurobi Optimiser, Version 7.5.1. For each vertex $v_i \in V$ we define integer variables $x_i = \{1, \ldots, k\}$ and binary variables y_i. The variable y_i is then equal to one if and only if vertex v_i is unhappy. The model is now:

$$\text{maximise} \quad n - \sum_{i=1}^{n} y_i \tag{10}$$

subject to:

$$x_i = c(v_i) \qquad \forall v_i \in V' \tag{11}$$

$$y_i \geq \frac{|x_i - x_j|}{n} \qquad \forall v_j \in \Gamma(v_i), \forall v_i \in V. \tag{12}$$

Here, Constraint (11) assigns all of the precolourings, while Constraint (12) sets $y_i = 1$ if and only if vertex v_i is unhappy. The objective function maximises the number of happy vertices.

In initial stages of our research, we also looked at a second, less compact, IP formulation which used binary variables x_{ij} (where $x_{ij} = 1$ if and only if vertex v_i was assigned to colour j) and variables $y_i \in R^+$ (which equalled one if and only if vertex v_i was unhappy). This was due to the following model.

$$\text{maximise} \quad n - \sum_{i=1}^{n} y_i \tag{13}$$

subject to:

$$x_{ij} = 1 \qquad \forall v_i \in V' : c(v_i) = j \tag{14}$$

$$\sum_{j=1}^{k} x_{ij} = 1 \qquad \forall v_i \in V \tag{15}$$

$$y_i \geq |x_{jk} - x_{ik}| \qquad \forall v_j \in \Gamma(v_i), \forall v_i \in V. \tag{16}$$

Here Constraint (14) specifies the precolourings, Constraint (15) ensures that only one colour is assigned to a vertex, and Constraint (16) sets $y_i = 1$ if and only if v_i is unhappy. For a small number of problem instances we found that this second model actually featured shorter run times than the first, though these patterns were not particularly clear. The second model was also seen to be far less reliable, with Gurobi often not producing an integer solution or completing its presolve routines within the imposed time limits (given below). All results in this paper therefore relate to the first model only.

For our trials, two types of problem instance were considered: random graphs and scale-free graphs. Random graphs are produced by starting with a set of n isolated vertices and then adding edges between each pair of vertices independently with a fixed probability p. This leads to a graph with approximately $\binom{n}{2}p$ edges and a binomial-shaped degree distribution with mean $(n-1)p$ and variance $(n-1)p(1-p)$.

In contrast to the unbiased method of random graph generation, scale-free graphs are based around the real-world idea of "preferential attachment" in that, when a vertex is added to a graph, it is more likely to be made adjacent to existing vertices that have high degrees. Scale-free graphs are therefore those whose degree distributions follow a power law, in which a small number of "hub" vertices usually feature disproportionately high degrees compared to the remaining vertices. Scale free graphs are known to model various real-work networks such as the World Wide Web, social networks, and the citation networks of academic papers [6].

In our experiments scale-free graphs were produced using the Barabási-Albert method [6], using a parameter $q \in \{0, 1, \ldots, n\}$. In our case we start with a complete graph $G = (V, E) = K_q$, comprising q vertices and $\binom{q}{2}$ edges. In each step a new vertex v is then added to G together with q edges that connect v to vertices already in G. This is done via a series of q roulette-wheel trials where, in each case, the probability $P(u, v)$ of adding the edge $\{u, v\}$ to E is calculated

$$P(u,v) = \begin{cases} \dfrac{\deg(u)}{\sum_{w \in (V - \Gamma(v))} \deg(w)} & \text{if } \{u,v\} \notin E \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

Here, $(V - \Gamma(v))$ denotes the set of vertices in G that are not yet adjacent to v. Vertices are added in this way until a graph with n vertices and $m = \binom{q}{2} + q(n-q)$ edges is formed. According to this method, a setting of $q = 0$ gives an empty graph on n vertices, $q = 1$ leads to graphs with no cycles (trees), and $q = n - 1$ or $q = n$ gives the complete graph K_n.

Finally, upon generation of a random or scale-free graph, k colours are assigned to a user-specified proportion of randomly chosen vertices, ensuring that each colour is used at least once. Note that we do not consider the subdivision of problems in our experiments here because we have found that the necessary H-U separating sets do not seem to naturally arise in graphs generated using these methods. They may occur more frequently with more targeted choices of precoloured vertices and/or using different graph topologies, however.

5.2 Identifying Hard Problems

To identify where difficult-to-solve instances of the MHV problem lie, graphs of $n = 1000$ vertices were produced using values of $k \in \{10, 50, 100, 250\}$. A large number of different settings for the proportion of precoloured vertices and graph density were then considered and, for each parameter combination, twenty separate instances were produced for both random and scale-free graphs. This resulted in more than $72,000$ problem instances in total. Our IP method was then executed on each of these graphs using a maximum run-time of 600 seconds. Note that our IP method was able to produce at least one integer solution for all of the graphs considered, but does not necessarily prove optimality.

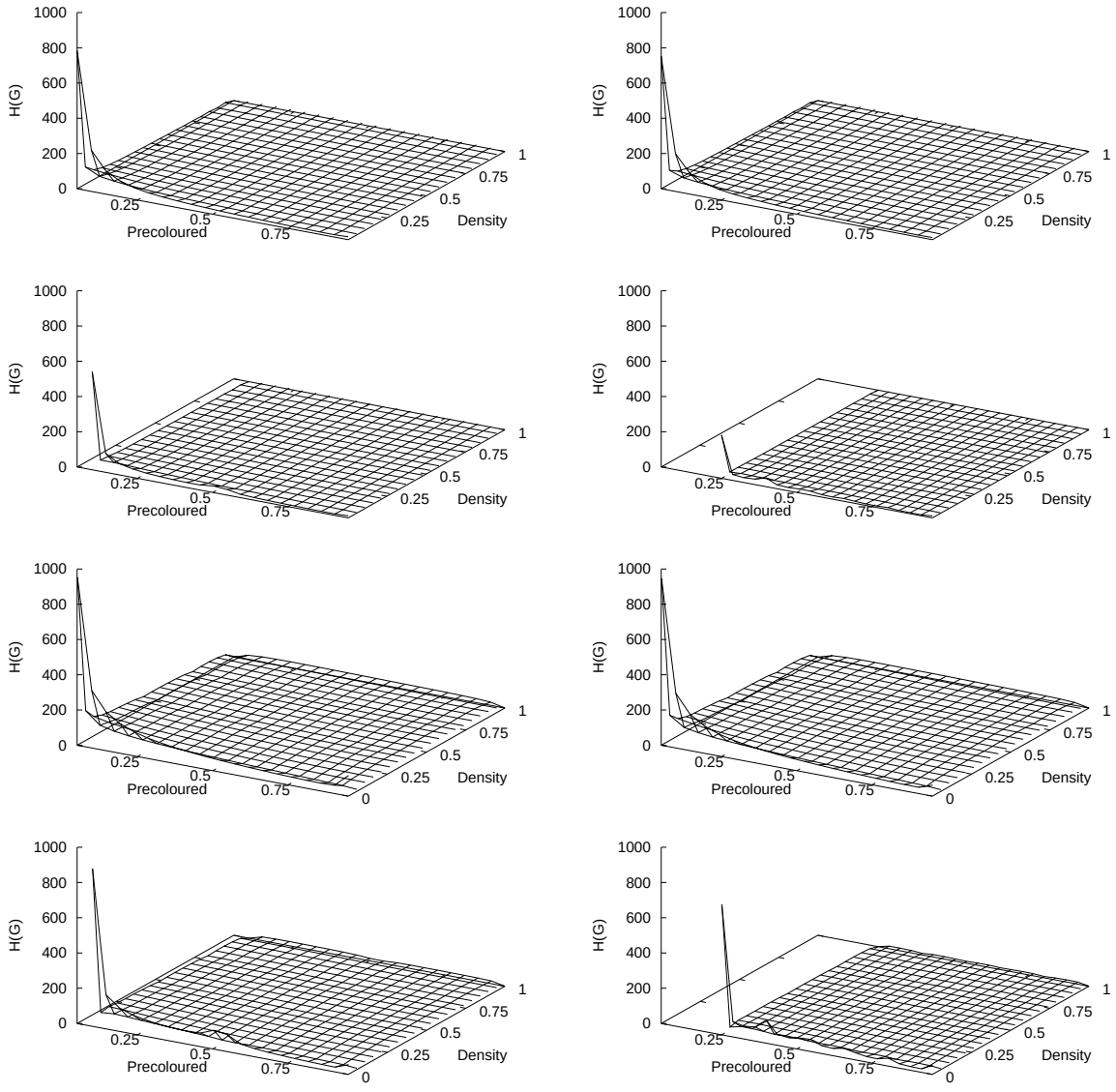

Figure 3: Number of happy vertices, $H(G)$, achieved by our IP method on problem instances with various proportions of precoloured vertices and graph densities. In order, these figures show the results for random graphs using $k = 10, 50, 100$ and 250, and then scale-free graphs using $k = 10, 50, 100$ and 250.

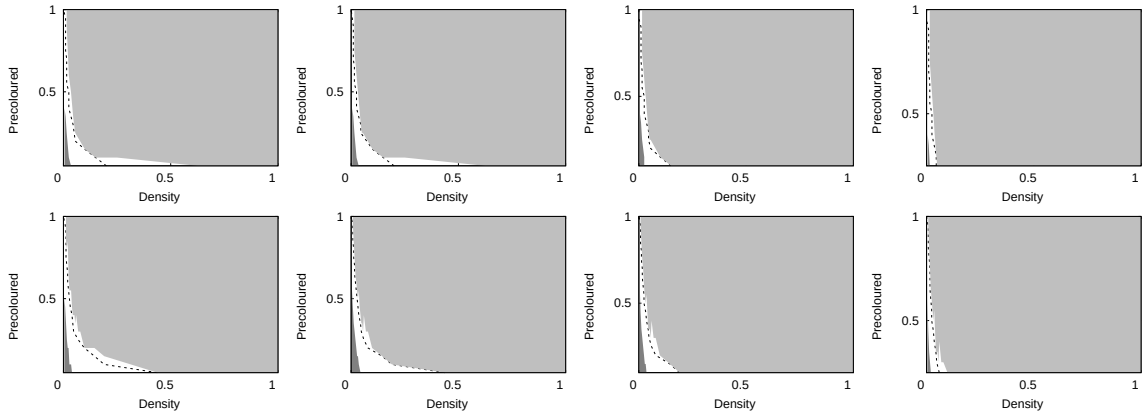

Figure 4: Dark grey areas indicate graphs where our IP method was not able to produce provably optimal solutions in all cases; light grey areas indicate graphs for which $H(G)^* = 0$; and areas to the right of the dotted line indicate graphs for which our generated $\bar{H}(G) = H(G)^*$. The top row shows random graphs, the bottom scale-free graphs, for $k = 10, 50, 100$ and 250 respectively. All points are calculated from twenty graphs produced at that location.

Our first set of figures uses the solutions produced by our IP method to examine how the number of happy vertices available in a problem instance alters for different graph densities and proportions of precoloured vertices. Figure 3 shows this for all four values of k using both random and scale-free graphs. Note that lower values are not present in the "precoloured" axes of $k = 100$ and $k = 250$ since, according to Definition 2, in this problem we are not permitted to have fewer precoloured vertices than available colours.

Figure 3 demonstrates that, for the majority of graphs, our IP method produces solutions that have no happy vertices. This finding is fairly obvious because in relatively dense graphs, vertices tend to have high degrees and are thus more likely to be adjacent to vertices of different colours. Similarly, when the proportion of precoloured vertices is high, this increases the chances of each free vertex being adjacent to at least two different colours, also ensuring that it cannot be happy. When happy vertices do exist, we also see that their numbers fall as k is increased. Again, this is fairly obvious when we consider that, with more colours, each vertex has a higher chance of being adjacent to two or more different colours, making it unhappy. As shown, these results are consistent across both random and scale-free graphs.

Of course, with our imposed time limit of 600 seconds the solutions produced by our IP method will not always be optimum, so we now consider the effects that different graph types have on the observed difficulty of a problem. These results are summarised in Figure 4 where we see that our IP method is only unable to solve these problems to optimality when the density is low, and/or the proportion of precoloured vertices is low. We also see that these patterns are very similar for random and scale-free graphs, and for different values of k.

Like Figure 3, Figure 4 also indicates the large number of graphs that feature no happy vertices ($H(G)^* = 0$). This fact was confirmed by our method of upper bound generation from Section 3 (which returned $\bar{H}(G) = 0$) and by our IP method, although the latter could sometimes take a few minutes to determine this, while the former was almost instantaneous. The figures also show where our upper bound from Section 3 has proved equal to the optimum solutions returned by our IP method, which includes some graphs for which $H(G)^* > 0$. We generally found, however, that this upper bound became less accurate for graphs with lower densities and lower proportions of precoloured vertices.

In Figure 5, we now focus more on low density graphs and examine the success rates and the CPU times required by our IP method. The success rates, indicated by the shading in the figure, show the proportion of runs where the algorithm was able to find the provably optimal solution within the 600 second time limit. The CPU time then indicates the average number of seconds required to find the optimum solution (unsuccessful runs are not considered in these averages).

The charts in Figure 5 confirm what we saw in Figure 4: that the most difficult problem instances are those with low densities and/or few precoloured vertices. For random graphs, these effects are fairly consistent, with denser graphs and/or increased numbers of precoloured vertices increasing the algorithm's success rate and reducing its required run times. Similar patterns also occur with the scale-free graphs with the notable exception of the least dense graphs. This is indicated in the figures by the very thin white strip running parallel to the "precoloured" axes, and the corresponding dips in CPU time. These particular graphs were generated using $q = 1$ (Section 5.1), resulting in trees with density $(\binom{q}{2} + q(n-q))/\binom{n}{2} = 0.002$. As noted in Section 1, it is known that the MHV problem is polynomially solvable on trees [4], and it seems here that our IP method is also able to consistently solve these problems to optimality, though the required run time may be significantly higher than more specialised algorithms.

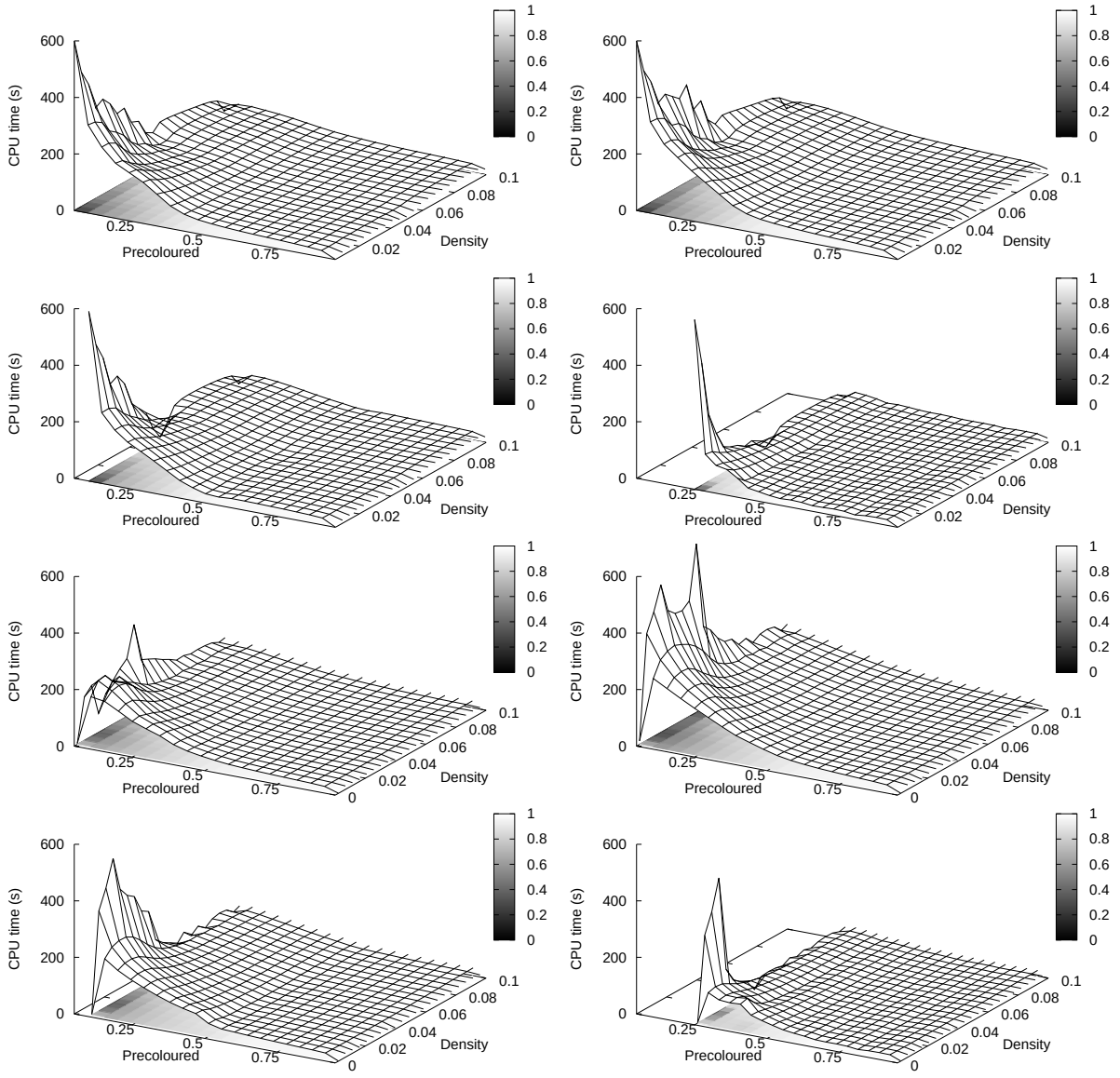

Figure 5: Success rates (shaded) and run times of graphs for various proportions of precoloured vertices and graph densities. In order, these figures show the results for random graphs using $k = 10, 50, 100$ and 250, and then scale-free graphs using $k = 10, 50, 100$ and 250.

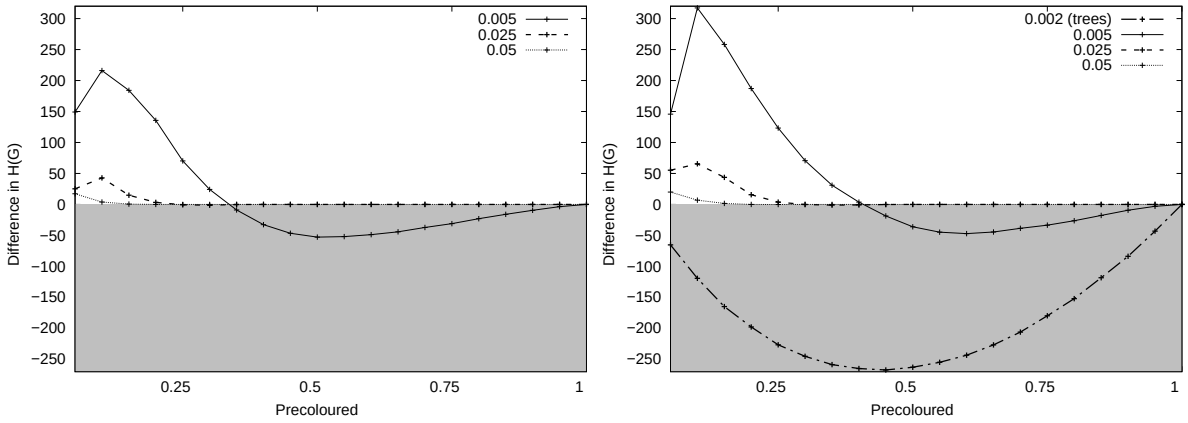

Figure 6: Comparison of the GREEDY-MHV and GROWTH-MHV algorithms for various densities and proportions of precoloured vertices, using random $k = 10$ graphs (left) and scale-free $k = 50$ graphs (right). Vertical axes are calculated as the $H(G)$ value returned from GREEDY-MHV minus the $H(G)$ value from GROWTH-MHV, averaged across 20 graphs. The unshaded area therefore indicates where GREEDY-MHV is superior on average, and the shaded area where GROWTH-MHV is superior.

5.3 Constructive Algorithm Comparison

We now consider the performance of the two constructive algorithms from Section 2. Figure 6 compares the number of happy vertices achieved by the two methods using $n = 1000$. For both random and scale-free graphs we see that the very basic GREEDY-MHV algorithm returns superior solutions when the number of precoloured vertices is low, and that these effects increase for sparser graphs. This stands to reason because under these conditions graphs will contain many LF-vertices, which are guaranteed to be happy in solutions produced by GREEDY-MHV. One notable exception to these patterns is shown in Figure 6 (right) for density 0.002. These graphs are scale-free trees, generated using $q = 1$. As a result, they contain many leaf vertices and only a small number high-degree "hub" vertices. Whenever a leaf vertex v is adjacent to a precoloured vertex in such graphs, colouring v with the same colour as its neighbour will necessarily make v happy. This approach is followed by GROWTH-MHV, but not by GREEDY-MHV, reducing the quality of its returned solutions.

We also compared the results of the two constructive algorithms against those of our IP method. For random graphs we found that GREEDY-MHV was able to produce superior solutions in approximately one quarter of trials when both graph density and the proportion of precoloured vertices was very low (values less than 0.03 and 0.25 respectively). Similar results were also observed with scale-free graphs, again with the exception of scale-free trees for the reasons noted above. In all other cases, the average number of happy vertices in solutions produced by our IP method was always equal or superior to those produced by GREEDY-MHV and GROWTH-MHV. Note, however, that the constructive algorithms are still very useful because they feature very low run times (for $n = 1000$, GREEDY-MHV completed in less than 0.01 seconds in all cases, while GROWTH-MHV completed in less than 1.2 seconds). They can, therefore, be used for quickly calculating lower bounds on $H(G)$, which might then form an additional constraint or starting solution for the IP method.

5.4 Scaling-up Issues and a Metaheuristic Approach

In this section we examine the influence that graph size has on the performance of our algorithms. As we shall see, our IP method experiences difficulties as the number of vertices n is increased; consequently, we will also propose an alternative metaheuristic algorithm that attempts to resolve some of these issues.

Our alternative algorithm is based on the Construct, Merge, Solve & Adapt (CMSA) methodology of Blum et al. [8]. The basic idea of CMSA is to consider the set C of all possible components that can be used for constructing candidate solutions. A single candidate solution S is then defined as a subset of these solution components in which the problem-specific constraints are also satisfied. During a run of CMSA, a subset of these components $C' \subseteq C$ is maintained, and an exact method is employed to produce candidate solutions that only use components currently residing in C'. The idea is that solving a problem with respect to C' requires less computational effort and, if the correct contents of C' are identified, high-quality solutions to the overall problem can be established more efficiently.

The pseudocode in Figure 7 describes our application of the CMSA method. The notation used here is chosen to be consistent with that used in [8]. For the MHV problem, the complete set of solution components is defined as $C = V \times \{1, \ldots, k\}$: that is, each component is a vertex/colour pair interpreted as an assignment of the vertex to the colour. A valid solution is then a subset of C in which each vertex is assigned to exactly one colour and where all

```
    CMSA (age_max, n_a)
(1) C' ← ∅
(2) age_c ← 0, ∀c ∈ C
(3) S_bsf ← GENERATE-HEURISTIC-SOLUTION
(4) while (time limit not exceeded) do
(5)     for i ← 1 to n_a do
(6)         S ← GENERATE-SOLUTION(S_bsf)
(7)         for all c ∈ S and c ∉ C' do
(8)             age_c ← 0
(9)             C' ← C' ∪ {c}
(10)    S'_opt ← APPLY-EXACT-SOLVER(C')
(11)    if S'_opt is better than S_bsf then
(12)        S_bsf ← S'_opt
(13)    ADAPT(C', S'_opt, age_max)
(14) return S_bsf
```

Figure 7: Pseudocode of the CMSA algorithm for the MHV problem.

the precolourings in the problem instance are obeyed. For example, the solution shown in Figure 1 can be written as: $\{(v_1, 1), (v_2, 1), (v_3, 1), (v_4, 1), (v_5, 2), \ldots, (v_8, 2)\}$

The CMSA algorithm is initialised in Steps (1) to (3) of the pseudocode. Here, the "ages" (explained below) of all components in C are set to zero, and the best-so-far solution S_{bsf} is generated. In our case S_{bsf} is found by executing both the GREEDY-MHV and GROWTH-MHV algorithms and taking the best of the returned solutions. Steps (4) to (13) then contain the main body of the algorithm. In each iteration n_a solutions are generated probabilistically using the GENERATE-SOLUTION procedure and, if not already present, the components of these solutions are added to the set C'. In Step (10) an exact solver is then used to solve the problem with respect to C'. In our case, we use the same IP formulation as before (Equations (10) to (12)) with the added restriction that each decision variable x_i can only assume values corresponding to the entries present in C'. Note that if $C' = C$ then an application of APPLY-EXACT-SOLVER is equivalent to our original IP method. Finally, upon completion of this step the best-so-far solution is updated (if appropriate) and the contents of C' are updated using the ADAPT procedure.

As indicated in the pseudocode, the intention of the GENERATE-SOLUTION procedure in Step (6) is to produce a solution S whose components can be added to the set C'. In our case this is achieved using S_{bsf} as a guide. Specifically, to produce S each vertex $v \in V$ is considered in turn and, if it is precoloured or has a degree of zero, it is automatically assigned to the same colour as the corresponding vertex in S_{bsf}; otherwise, a colour $j \in \{1, \ldots, k\}$ is selected randomly for v according to the probability:

$$P(v, j) = \frac{\phi(v, j)}{\deg(v)}, \tag{18}$$

where $\phi(v, j)$ gives the number of v's neighbours that are assigned to colour j in S_{bsf}. This probability function encourages vertices in S to assume colours that match the colour of their neighbours in S_{bsf}, but also allows new assignments to be made, therefore creating new components that can be added to C'.

Finally, the ADAPT procedure in Step (13) is used to periodically remove certain elements from C'. In our case, and as recommended in [8], this is achieved by first incrementing the age of all components in C' (i.e., $\text{age}_c \leftarrow \text{age}_c + 1,\ \forall c \in C'$), then setting the age of all components used in S'_{opt} to zero, and then removing any components from C' whose ages exceed the user-defined parameter age_{max}. This process prevents S' from becoming too large, but also encourages favourable components to remain in C' for subsequent cycles of the CMSA algorithm.

Table 1 now compares the performance of the constructive, IP and CMSA methods on a range of different problem sizes. In all cases random graphs were generated using values of $p = 5/(n-1)$, giving average vertex degrees of five, and 10% of the vertices were precoloured.[2] These can be considered "difficult-to-solve" problem instances according to our earlier results. Values of $k \in \{10, 50\}$ were also used, and twenty different graphs were generated in each case. For the IP and CMSA algorithms we used an extended run time of 3,600 seconds. For CMSA a maximum of 600 seconds was also permitted for each application of the exact solver, and parameter settings of $\text{age}_{\text{max}} = 3$ and $n_a = 10$ were used. These choices were made after preliminary experimentation.

The results in Table 1 indicate that, for these instances, around 55–60% of vertices can be happy. We also see that the IP method returns superior or equivalent results only for the smaller problem instances, with CMSA producing the best results for the remainder. Indeed the success rates for the IP method were only seen to be non-zero in just

[2]Recall from Section 5.1 that, for random graphs, p gives the probability of two vertices being adjacent.

		Upper Bounds (Mean ± SD)		$H(G)$ (Mean ± SD)[a]		
	n	$\bar{H}(G)$	UB[b]	Constructive[c]	IP Method	CMSA
$k = 10$	250	74.040 ± 2.1	60.760 ± 2.3	60.720 ± 2.4	**60.760 ± 2.3**	60.740 ± 2.3
	500	74.810 ± 1.4	63.050 ± 1.8	60.860 ± 2.5	60.890 ± 2.4	60.890 ± 2.4
	750	74.987 ± 0.9	65.027 ± 1.6	60.253 ± 1.3	60.287 ± 1.3	60.287 ± 1.3
	1000	75.000 ± 1.0	65.550 ± 1.1	59.780 ± 1.2	59.860 ± 1.2	59.860 ± 1.2
	2000	74.950 ± 0.9	66.780 ± 1.5	59.685 ± 1.1	59.698 ± 1.1	**59.720 ± 1.1**
	3000	74.998 ± 0.7	67.018 ± 1.0	59.518 ± 0.7	59.540 ± 0.7	**59.553 ± 0.7**
	4000	75.046 ± 0.6	70.720 ± 1.0	59.261 ± 0.8	59.053 ± 0.8	**59.296 ± 0.7***
	5000	74.886 ± 0.4	70.465 ± 1.0	58.928 ± 0.6	58.698 ± 0.6	**58.969 ± 0.6***
	7500	74.848 ± 0.4	70.723 ± 0.8	58.847 ± 0.6	58.680 ± 0.6	**58.885 ± 0.6***
	10000	74.080 ± 0.3	71.891 ± 6.2	57.918 ± 0.5	57.769 ± 0.6	**57.929 ± 0.5***
$k = 50$	500	74.310 ± 1.4	63.380 ± 2.4	56.180 ± 2.3	**56.260 ± 2.3**	56.250 ± 2.3
	750	74.533 ± 1.0	64.833 ± 1.7	56.780 ± 1.6	56.827 ± 1.6	56.827 ± 1.6
	1000	74.575 ± 1.0	64.955 ± 1.3	56.515 ± 1.1	56.585 ± 1.1	56.585 ± 1.1
	2000	74.453 ± 0.9	65.990 ± 1.4	56.358 ± 1.1	56.385 ± 1.1	**56.413 ± 1.1***
	3000	74.515 ± 0.7	66.190 ± 1.0	56.337 ± 0.8	56.350 ± 0.8	**56.392 ± 0.7**
	4000	74.583 ± 0.6	67.871 ± 1.5	56.200 ± 0.7	56.001 ± 0.8	**56.253 ± 0.7***
	5000	74.405 ± 0.4	69.014 ± 0.9	55.869 ± 0.7	55.808 ± 0.7	**55.930 ± 0.7***
	7500	74.341 ± 0.4	69.434 ± 0.7	55.911 ± 0.6	55.653 ± 0.5	**55.959 ± 0.6***
	10000	73.579 ± 0.3	69.851 ± 1.7	54.909 ± 0.4	54.760 ± 0.5	**54.918 ± 0.4***

[a] Asterisks indicate a significant difference between the best two methods at the 0.1% level according to a Wilcoxon signed rank test.
[b] Upper bound returned by the IP method within the 3,600 second time limit.
[c] Best solution found in runs with both GREEDY-MHV and GROWTH-MHV.

Table 1: Results achieved by the constructive, IP, and CMSA methods using random graphs with average degrees of five and 10% precoloured vertices. All figures are the mean and standard deviation from runs across twenty problem instances, expressed as percentages on the number of vertices n.

two classes of graph;[3] consequently, we are not able to definitively state the optimal number of happy vertices for the remaining instances at this point. Interestingly, for instances with 4,000 or more vertices, Table 1 indicates that the constructive algorithms also produce better solutions than the IP method. This further highlights the IP method's difficulties with these large problem instances. That said, the upper bounds returned by the IP solver at the end of execution are still more accurate than the bound of $\bar{H}(G)$ (Section 3.2) with these instances.

6 Conclusions and Further Work

This paper has analysed a number of features of the maximum happy vertices (MHV) problem, including methods for generating bounds and for breaking up problems. We have demonstrated empirically that difficult-to-solve problems only really occur when graph density and/or the proportion of precoloured vertices is low, which results in graphs with higher numbers of potentially happy vertices. These patterns have been observed to be quite consistent across different values of k and for both random and scale-free topologies.

Although our IP method is effective at solving many instances to optimality, it also suffers from scaling issues. As a result, a CMSA approach has also been suggested that achieves significantly better performance with larger graphs. We anticipate that other approaches—particularly bespoke metaheuristics or improved IP formulations—will show further promise in this regard.

One of the general difficulties of identifying polynomially solvable instances of the MHV problem is that, for any particular graph topology, various different combinations of vertices can be precoloured, each of which will lead to a differently structured problem instance. This is in contrast to the more well-known (vertex) graph colouring problem, where results are known for many different graph types [12]. To illustrate the effects that different choices of precoloured vertices can have on a problem, a small number of additional experiments were conducted on twenty scale-free graphs generated using $n = 1000$ and $q = 4$. In the first set of trials, the 5% of vertices with the highest degrees (the "hubs") were precoloured; in the second set, the 5% of lowest-degree vertices were precoloured. For $k = 10$, our IP method was not able to solve any of these instances to optimality; however, when the high-degree vertices were precoloured, approximate solutions with an average of just 226.4 happy vertices resulted. In contrast, for the second set this rose to 818.1 happy vertices. For $k = 50$ the corresponding figures were 191.3 and 789.5 happy vertices. Clearly then, even with the same graphs, the choice of precoloured vertices can have a large effect on the types of solution that are available.

In many real world situations, we may also choose to consider various generalisations of the MHV problem. For example, we can easily modify Definition 1 to say that a vertex v is happy if and only if $\min(\deg(v), l)$ of its

[3] Specifically: $n = 250$, $k = 10$ (where all instances were solved to optimality in an average of 69 seconds) and $n = 500$, $k = 10$ (where 20% of instances were solved to optimality in an average of 2,328 seconds).

neighbours are assigned to the same colour, where l is a parameter specified as part of the problem. For $l \geq \Delta$ we get the MHV problem itself, whereas $l = 0$ leads to a trivial problem (all solutions have n happy vertices). However, we are not currently aware of other values for l that are polynomially solvable. If we consider $l = 1$, for example, it is clear that if we have a u, v-path whose vertices are all assigned to the same colour, then all vertices in this path will be happy. An appropriate algorithm might therefore exploit this feature by acting as follows.

1. Let v be a vertex assigned to colour i, and let G' be the subgraph induced by vertices in G that are coloured with i, or that are uncoloured.
2. Starting from v, identify a spanning tree T of G' and assign all uncoloured vertices in T to colour i.
3. If G contains any remaining uncoloured vertices, return to Step 1; otherwise end.

Other interpretations of the MHV problem are also plausible, including those where we might wish to balance the number of vertices per-colour; decentralised problems, where individual vertices are only aware of colours assigned to a subset of the vertices (such as their neighbouring vertices); and problems where all coloured vertices need to be happy, but where we allow other vertices to remain uncoloured in a solution. Each of these variants is worthy of further research.

Acknowledgements

This research was partially supported by the Cardiff University International Collaboration Seedcorn Fund. It was also supported in part by the Monash eResearch Centre and eSolutions-Research support services through the use of the MonARCH HPC cluster. The authors are also grateful for the comments of the reviewers, which helped to improve the paper.

References

[1] Algorithm source code. `http://www.rhydlewis.eu/resources/happyalgs.zip`. Accessed 2018-11-14.

[2] Problem instance generator. `http://www.rhydlewis.eu/resources/happygen.zip`. Accessed 2018-11-14.

[3] A. Agrawal. On the parameterized complexity of happy vertex coloring. In L. Brankovic, J. Ryan, and W. Smyth, editors, *Combinatorial Algorithms. IWOCA 2017*, volume 10765 of *Lecture Notes in Computer Science*, pages 103–115. Springer, Cham., 2018.

[4] N. Aravind, S. Kalyanasundaram, and A. Kare. Linear time algorithms for happy vertex coloring problems for trees. In V. Mäkinen, S. Puglisi, and L. Salmela, editors, *Combinatorial Algorithms: IWOCA 2016*, volume 9843 of *Lecture Notes in Computer Science*, pages 281–292. Springer Cham., 2016.

[5] N. Aravind, S. Kalyanasundaram, A. Swami Kare, and J. Lauri. Algorithms and hardness results for happy coloring problems. *CoRR*, abs/1705.08282, 2017.

[6] A. Barabási. *Network Science*. Cambridge University Press, 2016.

[7] M. Bellows and J. Luc Peterson. Finding an optimal seating chart. *Annals of Improbable Research*, 2012.

[8] C. Blum, P. Pinacho, and J. López-Ibáñez, M. amd Lozano. Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. *Computers and Operations Research*, 68:75–88, 2016.

[9] M. Carter, G. Laporte, and S. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47:373–383, 1996.

[10] B. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. Probability and Statistics. John Wiley & Sons, 5th edition, 2011.

[11] A. Kanevsky. Finding all minimum-size separating vertex sets in a graph. *Networks*, 23:533–541, 1993.

[12] R. Lewis. *A Guide to Graph Colouring: Algorithms and Applications*. Springer International Publishing, 2016.

[13] R. Lewis and F. Carroll. Creating seating plans: A practical application. *Journal of the Operational Research Society*, 67(11):1353–1362, 2016.

[14] R. Lewis and J. Thompson. Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, 240:637–648, 2015.

[15] A. Li and P. Zhang. Algorithmic aspects of homophyly of networks. *Theoretical Computer Science*, 593:117–131, 2015.

[16] B. McCollum, A. Schaerf, B.. Paechter, P. McMullan, R. Lewis, A. Parkes, L. Di Gaspero, R. Qu, and E. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.

[17] N. Misra and I. Vinod Reddy. The parameterized complexity of happy colorings. In L. Brankovic, J. Ryan, and W. Smyth, editors, *Combinatorial Algorithms. IWOCA 2017*, volume 10765 of *Lecture Notes in Computer Science*, pages 142–153. Springer, Cham., 2018.

[18] P. Zhang, T. Jiang, and A. Li. Improved approximation algorithms for the maximum happy vertices and edges problems. In D. Xu, D. Du, and D. Du, editors, *Computing and Combinatorics: 21st International Conference, COCOON 2015, Beijing, China*, pages 159–170. Springer, 2015.